Lista zadań nr 10

Zadanie 1. (2 pkt)

Zaimplementuj w języku IMP program rozkładający podaną liczbę na czynniki pierwsze. Na przykład, jeżeli użytkownik poda liczbę 220, program powinien wyświetlić następujące liczby.

```
2
2
5
11
```

Zadanie 2. (2 pkt)

Rozbuduj interpreter języka FUN z drugiej tercji wykładu o mutowalny stan, podobny do tego, który jest dostępny w OCamlu. W rozszerzonym języku powinieneś być w stanie napisać program podobny to funkcji `next` z ostatniego wykładu:

```
let next =
  let x = ref 0 in
  fun y -> x := !x + 1; !x
in
next 0 + next 0
```

W tym celu rozszerz język o następujące trzy nowe konstrukcje.

- `ref` e — oblicza wyrażenie e do wartości v i tworzy nową komórkę pamięci l przechowującą v. Wartością całego wyrażenia jest komórka l.
- `!`e — oblicza wyrażenie e do wartości, która powinna być pewna komórką pamięci l. Wartością całego wyrażenia jest wartość przechowywana w l.
- e_1 `:=` e_2`;` e_3 — oblicza wyrażenie e_1 do pewnej komórki l, następnie oblicza e_2 i wynik przypisuje do l. Na koniec oblicza e_3, a otrzymana wartość jest wartością całego wyrażenia.

Zaproponuj składnię abstrakcyjną dla tych konstrukcji i rozbuduj parser i lekser. Następnie rozbuduj interpreter, o obsługę stanu w sposób *metacykliczny*, czyli użyj typu `'a ref` do zaimplementowania stanu. Oczywiście trzeba rozszerzyć typ wartości o nowy konstruktor reprezentujący komórki pamięci. Powinieneś zrobić to w następujący sposób.

```
type value =
  ...
  | VRef of value ref
```

Zadanie 3. (2 pkt)

Dodaj do języka IMP z wykładu lokalne definicje zmiennych. W tym celu zmodyfikuj składnię tak, by bloki instrukcji zaczynały się listą deklaracji zmiennych lokalnych dla danego bloku:

```
stmt:
  ...
  | BEGIN; vs=list(var_decl); s=list(stmt); END { Block(vs, s) }
  ...
  ;.
```

Następnie zmodyfikuj *metacykliczny* interpreter tak by wspierał nową cechę języka. Oczywiście przesłanianie zmiennych powinno działać zgodnie z oczekiwaniami. Na przykład poniższy program powinien wyświetlić 42.

```
begin
  var x
  x := 42
  begin
    var x
    x := 13
  end
  write x
end
```

Zadanie 4. (3 pkt)

Powtórz poprzednie zadanie, tym razem rozbudowując czysto-funkcyjny interpreter. W tym zadaniu nie wolno używać mutowalnego stanu! Twoje rozwiązanie powinno działać w stałej pamięci, a dokładniej, ilość potrzebnej pamięci może

zależeć od rozmiaru interpretowanego programu, ale nie może zależeć od czasu jego działania.

Wskazówka: zauważ, że zmienne lokalne mają dyscyplinę stosu. Rozdziel koncepcję sterty od środowiska: sterta będzie stosem wartości, zaś środowisko będzie mapować nazwy zmiennych na miejsca na stosie.

Zadanie 5. (2 pkt)

Rozwiąż ponownie zadanie 2, tym razem używając nie używając mutowalnego stanu. Czy tym razem możemy polegać na dyscyplinie stosu? Nie przejmuj się, jeżeli Twoje rozwiązanie nie działa w stałej pamięci.

Zadanie 6. (1 pkt)

Rozszerz parser języka IMP z preinkrementacją o obsługę nowej konstrukcji (składnia konkretna dla preinkrementacji to ++x). Dodatkowo rozszerz język o operację postinkrementacji (x++), która działa podobnie do preinkrementacji, ale z tą różnicą, że wartością wyrażenia x++ jest wartość zmiennej x przed inkrementacją.